FROM REACTIVE MULTI-AGENTS MODELS TO CELLULAR AUTOMATA

Illustration on a Diffusion-Limited Aggregation Model

Keywords: Reactive multi-agent systems, cellular automata, influence-reaction model, diffusion-limited aggregation

Abstract: This paper deals with the synchronous implementation of situated Multi-Agent Systems (MAS) in order to have no execution bias and to allow their programming on massively parallel computing devices. For this purpose we investigate the translation of discrete MAS into Cellular Automata (CA). Contrarily to the sequential scheduling generally used in MAS simulations, CA is a model for massively parallel computing where the updating of the components is synchronous. However, CA expressivity is limited and not adapted to build models where independent entities may move and act on neighbor cells. After illustrating these issues on a simple example, we propose a generic method to translate discrete MAS into CA, called transactional CA. Our approach consists in translating MAS specified with the influence-reaction model into a transactional CA.

1 INTRODUCTION

Multi-agents systems (MAS) are widely used for modeling systems where autonomous entities, the *agents*, move in a virtual space, the *environment*, and act on it. Numerous simulators and platforms have been developed to simulate such systems. However, in these tools, the updating of the agents is often left as a hidden procedure, on which the user has no control. The most common updating procedure is the sequential procedure: agents are updated one after the other with an order fixed in advance (often randomly). It is a well-known problem that such scheduling is a potential source of biases, *i.e.*, it may introduce causalities that were not designed by the user but come only from the simulating tool. By contrast, Cellular automata (CA) are a well-known model of massively parallel computing devices where the updating of the components is synchronous: all the cells are updated at once without any priority between them. The advantage of using the CA formalism is simplicity: it involves static homogeneous computing units that are regularly arranged in space. The drawback of expressing a model with cellular automata appears when one needs to build models with pseudo-independent entities that may move and act on neighbor cells. Indeed, in CA, a cell cannot write or move directly on its neighbor cells, whereas such an ability is usually required to express a MAS model. This paper investigates the translation of discrete MAS models into CA, which is illustrated on a simple example. The interest is twofold: (1) to have a synchronous execution of agents and thus to reduce bias due to the update, and (2) to ease the programming of MAS on massively parallel computing devices such as FPGAs or GPUs.

The purpose of our research is to find a method to translate ”the language of multi-agents” into ”the language of cellular automata”. In this article, we propose to take advantage of both contexts, the high expressiveness of a MAS specification and the simplicity of a CA implementation. We aim at developing a framework where MAS, that are simply described through the separate specifications of the local agent behaviors and the environment dynamics, are automatically translated as a uniform transition function of a CA.

This article is organized as follows: In section 2, we discuss the relations between CA and MAS approaches. Section 3 introduces the concept of *transactional CA*, starting from the study of a paradigmatic example of a MAS model, namely the Diffusion-Limited Aggregation model. Section 4 proposes the

first step of a formal description allowing the generic coding of a reactive MAS model into a transactional CA. We finally conclude with discussions on related and future works.

2 MAS *versus* CA

At first sight, the two formalisms look very similar and are often confused. One may find several works where the names "cellular automata" and "multi-agent systems" are used without distinction. This is easily understandable since CA are often used to model the environment of a MAS, and, reciprocally, one may see a CA as a particular kind of MAS where agents do not move.

We now clarify the differences between CA and MAS in the context of research. We compare them at two levels: (1) the modeling level: *What in a model makes CA or MAS more suitable to express it?* (2) the simulation level: *How intuitive is the implementation of CA and MAS?*

MAS and CA, as modeling tools. In their definition, CA are uniform objects: there is a *unique* neighborhood shape for each cell and a *unique* transition function. As a consequence, CA are fitted to model phenomena that involve *homogeneous* spaces; CA have been used for example to model physical systems (Chopard and Droz, 2005), biological systems (Deutsch and Dormann, 2005), spatially embedded computations (Adamatzky, 2001), etc. Note that it is always possible to take into account inhomogeneities, for example by encoding the heterogeneity in the cell states, but this is generally not straightforward to do so.

MAS are preferred for expressing an heterogeneous population of entities. They necessitate to make a distinction between the agents' behaviors and the *environment* where they are embedded (Ferber, 1999). This distinction allows us to focus on the specification of particular and localized events, namely the agents *actions*. They offer a methodology for designing systems, at the level of algorithms, programming languages, hardware, etc. Examples of MAS applications range from the simulation of natural systems, from ants (Resnick, 1994) to human behaviors (Regelous, 2004), to the design of massively distributed software and algorithms like web-services, peer-to-peer technologies, etc. Nevertheless, we must note that contrarily to CA, no universal definition of MAS has been accepted so far. From the modeling point of view, *translating MAS in the cellular automata formalism has (at least) the advantage of fixing the mathematical expression of the model and removing ambiguities of formulation.*

MAS and CA, as simulation tools. The key characteristic of complex systems is the difficulty, if not the impossibility, of inferring their global behavior from the local specification of the interactions. Few mathematical tools are available to predict the evolution of complex systems in general, more especially those which involve self-organization. This gives to simulation a central role to find the mechanisms that explain how complexity emerges from simple local interactions. We thus have to pay attention to the quality of simulations and to detect ambiguities that may be hidden in the way they are implemented.

The agent-based programming style is somehow intuitive and natural as the programmer takes the point of view of the agent. There is a form of anthropomorphism that makes MAS programming particularly attractive. Nevertheless, we emphasize that once all the agents behaviors are individually specified, there are still many ways to make the agents interact and play together in the environment. The implementation of such systems raises many questions, like assessing the importance of the synchronicity in simulations: are the agents updated all together or one after the other? The design of spatially-extended computing devices will require to imagine a new type of computer science, where the computations do not *necessarily* rely on the existence of a synchronization between the components.

By contrast with MAS, CA lead to shift the programmer's point of view from the "eyes" of the agents to their environment. The benefits of this shifting effort are twofold: (1) the CA formalism forces the programmer to solve conflicts between concurrent agents actions at the elementary level of the cell and forbids the use of any global procedure. (2) As a consequence, the implementation on massively distributed devices is easy. Indeed, CA provide the programmer a cell-centered programming style where the set of cells represents computing units that are regularly organized. Recent works have shown that it is possible to have a good efficiency by using parallel architecture to run CA simulations for FPGA (Halbach and Hoffmann, 2004) and for GPU (Permalla and Aaby, 2008). In other words, *CA provide an easy-to-implement framework, but expressing the local rule necessitates a method to "blend" the different components of a complex system.*

3 THE DLA EXAMPLE AS A STARTING POINT

In this section, we introduce our approach through the translation of a simple MAS model into an original kind of CA, called *transactional CA*. For this purpose, we focus on the CA encoding of a *diffusion-limited aggregation* (DLA) system. This example presents a good trade-off between the simplicity of description and the richness of problems risen by this coding.

The DLA model was introduced to study physical processes where diffusing particles, following a Brownian motion, aggregate (Witten and Sanders, 1981): for instance, zinc ions aggregate onto electrodes in an electrolytic solution. This process leads to interesting self-organized dendritic fractal structures. Different models of DLA have been proposed; we consider in this article that particles stick together forever and that there is no aggregate formation between two mobile particles.

3.1 MAS Specification of the DLA

The MAS specification of the DLA model describes separately the *agents* and the *environment* where they evolve:

The environment is a 2D finite and toric square grid composed of elements called *patches*. The *exclusion principle* holds: *i.e.*, there cannot be more than one agent on each patch of the grid.

The population of agents, denoted by $\mathcal{A}$, is composed of the particles. Each particle a of $\mathcal{A}$ is localized on a cell ρ_a of the environment and is characterized by a state σ_a: a particle is either Fixed or Mobile.

The initial configuration of the system is composed of a population of Mobile particles and some Fixed particles called the *seeds*. The expected behavior is the aggregation of the Mobile particles to build dendrites from the seeds.

We propose to formulate the agent dynamics using the usual *perception-decision-action* cycle (Brooks, 1990). We first describe the perception and action abilities of an agent. The *perception* consists of two functions:

- Γ_1 returns true if the agent perceives a Fixed *neighbor* particle, and false otherwise;
- Γ_2 computes the set of *directions* that lead to empty *neighbor* patches.

The neighborhood referred in these perceptions corresponds to the four closest positions of ρ_a following North, South, East and West directions. The *set of actions* is:

- Diffuse(d): move following direction d;
- Aggregate: change to the Fixed state;
- Stay: do nothing

Let $\mathcal{U}(S)$ denote the operation of selecting one element in a finite set S with uniform probability, the *decision process* returns an action as a function of the agent perceptions:

$$\begin{array}{llll} \text{if} & \Gamma_1 & \text{then} & \text{Aggregate} \\ \text{else if} & \Gamma_2 \neq \emptyset & \text{then} & \text{Diffuse}(\mathcal{U}(\Gamma_2)) \\ \text{else} & & & \text{Stay} \end{array} \quad (1)$$

3.2 CA Expression of the DLA Model

We now reach the core of the problem. We first discuss about implementing the agent motion within a synchronous computational model. We then propose our solution, called *transactional CA*, and we finally illustrate it on the DLA example.

The Synchrony Paradox. In the MAS style of programming, emphasis is put on the agents local behaviors. Classically, to avoid collisions between mobile particles, mobile particles are introduced one after the other, or in some cases, are introduced simultaneously but updated one after the other using a scheduler. However, two objections can be raised:

1. The implementation of this sequential updating on a massively distributed computing device is not impossible, but it requires the introduction of complex procedures to synchronize the different schedulers.
2. The use of a scheduler introduces an external form of causality that was not specified in the original DLA formulation. This may induce a bias in the formation of dendritic patterns, especially when the density of mobile particles is high.

By contrast, the framework of CA demands an early resolution of the conflicts created by simultaneous moves to a given patch. To achieve that, we propose to establish a dialog between cells.

Transactional CA. A particle move requires a *source* cell (that contains a particle at time t) and a *target* cell (that will contain the particle at time $t+1$). We propose to elaborate a three-step *transactional* process where cells negotiate their requirements:

1. *Request*: *source* cells express their needs to their neighbors.

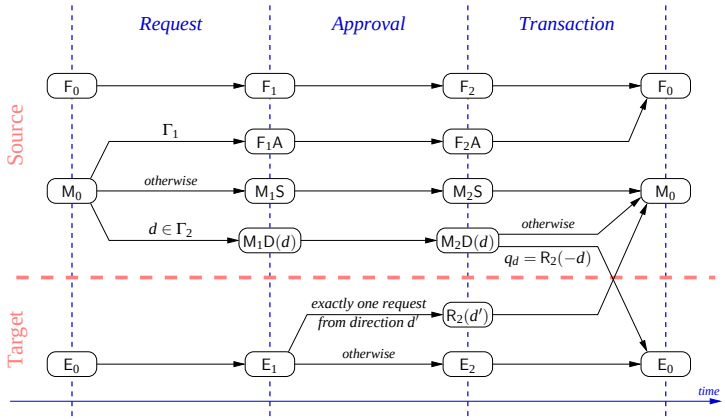

Figure 1: DLA local evolution rule within a transactional CA. This graph shows the local evolution of a cell from a state to another depending on its neighborhood state. Explanations are given in the text.

2. *Approval-rejection*: *target* cells accept or not their neighbors requirements; this decision is done with respect to an exclusion principle policy (for example, an empty cell is an available target iff there is exactly one particle requesting to move to this cell).
3. *Transaction*: *sources* and *targets* separately evolve.

DLA Transaction Model. Figure 1 shows with a graph the local transition function of a transactional CA capturing the agent-based specification of the DLA given in section 3.1. On this graph, nodes represent the different states of the CA (states E_0, F_0 and M_0 are given twice to clarify the figure) and the arrows specify transitions between states. States are distributed

- *vertically*, to segregate the behaviors of sources and targets, and
- *horizontally*, to distinguish the three steps of a transactional CA.

At the beginning, the cells are either empty or contain a particle which is either fixed or mobile: three states are used E_0, F_0 and M_0.

- The request transition consists in deciding an action for each M_0 cell: depending on the perceptions, a mobile particle either aggregates (state F_1A), or requests diffusion following a direction d (state $M_1D(d)$), or stays at the same position (state M_1S).
- During the approval step, empty cells E_1 decide, by reading their neighbors requirements, if they remain empty (state E_2) or become receptors of particles moving from a direction d' (state $R_2(d')$).
- Finally, the transaction is computed: receptors become particles, moving particles with a receptor target (*i.e.*, when the state q_d of the pointed cell is $R_2(-d)$, where $-d$ denotes the direction opposite to d) become empty, and aggregating particles become fixed. Other cells remain in their initial state.

Figure 2 presents simulations of the previous described DLA model in two simulations frameworks. On the first line, simulations were obtained using a classical sequential framework based on a scheduler, on the second line, we display simulations of our synchronous transactional CA. The same initial configuration, given on the left column, was used on both platforms. It consists of a 100x100 grid where seeds are localized on the boundaries and where mobile particles are gathered in a 40x40 central square. Both systems exhibit the same qualitative behavior, as seen on the right column of Figure 2. However, further studies on the dendrites distribution or on the mean time required to reach a fixed point would be needed to assess the differences between the two approaches. To compare the time scales of the two systems, we define a *simulation time step* as: (a) the three substeps of the transactional CA and (b) the update of all the agents in the sequential framework. We observe that the dissolution of the initial square is slower in the synchronous CA than it is in the MAS model with a sequential updating (see the middle column of Figure 2). A simple explanation of this phenomenon is that an asynchronous update allows a particle to move to a just evacuated patch *during a simulation time step*, while the synchronous update forbids this behavior.

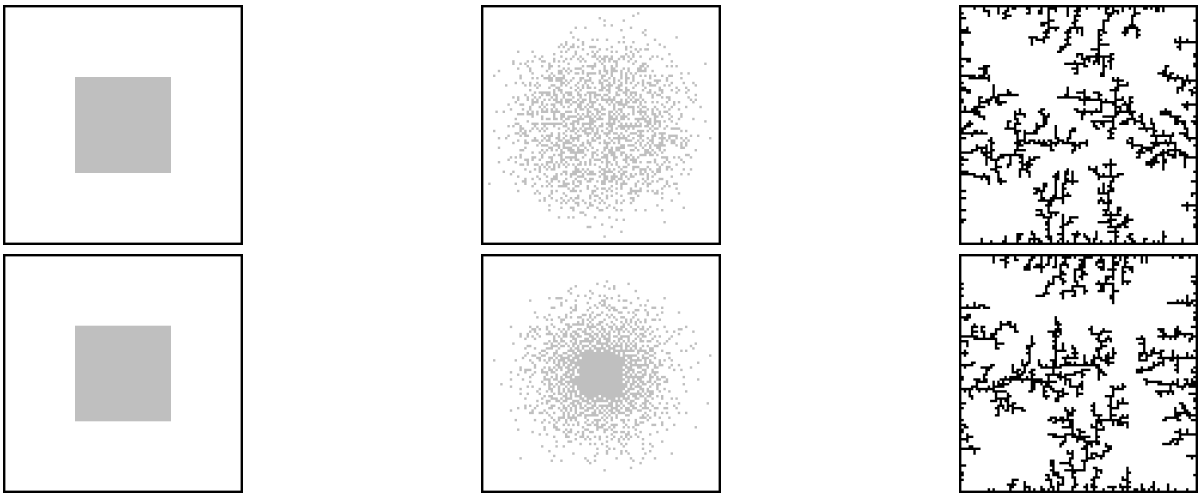

Figure 2: DLA Simulations: from left to right, the initial state, the state after 130 simulation time steps and the fixed point, with in black the fixed particles and in gray the mobile particles. The first line was obtained using the sequential simulation tool TurtleKit (Michel et al., 2003), the second was obtained using the CA simulation tool FiatLux (Fatès, 2008).

4 TOWARDS A GENERALIZATION

In this section, we investigate how a generic method could be developed to automatically translate the specification of a MAS into the transition function of a transactional CA. Of course, reducing a MAS to a CA enforces some restrictions on what can be described. More especially, in order to respect the finiteness of CA, we assume that MAS are discrete and finite systems: *i.e.*, the environment is a *discrete* and *regular* grid where a *finite* number of agents are localized on specific parts of this grid (they do not have continuous coordinates).

Our approach is based on the use of the formal *influence-reaction* model (Ferber and Muller, 1996) to describe a MAS. In fact, the three steps of the transactional CA are similar to the three steps of influence-reaction: (1) agents produce influences that are attempts of actions, (2) influences are combined to avoid conflicts between the corresponding actions, and (3) the environment is updated with respect to the combined influences. In the following, we formally introduce the influence-reaction model and we finally give the first step of a formal description of an *automatic* translation of an influence-reaction based specification into a transactional CA.

4.1 Influence-Reaction Model

On the opposite of the CA, there is no unique formal description of MAS models. Nevertheless, there exists some generic models that focus on specific kinds of MAS (*e.g.*, logic MAS, communicating MAS, etc.). For the sake of clarity, we only consider *situated* MAS that deal with *discrete* environment and *reactive* agents. The term "situated MAS" relates to systems where agents are embedded in a "physical" environment.

In this context, we focus on the *influence-reaction model* that is dedicated to the formal description of situated MAS, allowing, in particular, the simulation of simultaneous actions (Ferber and Muller, 1996). In this model, agents release *influences* that will induce *reactions* of the environment. An influence corresponds to attempting an action. The reaction consists in combining the different influences in order to realize the corresponding actions. This principle is inspired by physics where entities react because some forces act on them. Like forces, influences can be combined. For instance, an influence may be an attempt to pull a door. If two agents simultaneously perform this influence with the same intensity from the opposite sides of a door, the combination of both influences vanishes and the resulting action is null.

We assume here that the combination of influences can be computed *locally*. In other words, the evaluation of the combination function can be distributed on each patch of the environment. The dynamics of the three steps of the influence-reaction model may be de-

scribed as follows:

1. Each agent a of $\mathcal{A}$ separately computes, as a function f_a of its current state σ_a^t and of its perceptions Γ_a, its new state σ_a^{t+1} and the associated set of influences I_a. We denote by $\mathcal{I}$ the set of all the possible influences, $\Sigma_{\mathcal{A}}$ the set of the agent states.
2. Let I_ρ denote all the influences produced by the agents of $\mathcal{A}$ that could affect the patch ρ of the environment (we denote by $\mathcal{P}$ the set of all the environment patches). Each patch ρ separately computes the set $\prod I_\rho$ of combined influences affecting the patch ρ.
3. Finally, for each patch ρ of the environment, the new state σ_ρ^{t+1} of ρ is computed as a function $f_{\mathcal{E}}$ of the current position state σ_ρ^t with respect to the set of influences $\prod I_\rho$. We denote by $\Sigma_{\mathcal{E}}$ the set of the patch states.

Using these notations, the influence-reaction dynamics may be formally summarized by the following two equations:

$$\langle \sigma_a^{t+1}, I_a \rangle = f_a(\sigma_a^t, \Gamma_a) \quad a \in \mathcal{A} \qquad (2)$$

$$\sigma_\rho^{t+1} = f_{\mathcal{E}}(\sigma_\rho^t, \prod I_\rho) \quad \rho \in \mathcal{P} \qquad (3)$$

4.2 Generation of a Transactional CA

Formally speaking, a CA is a 4-tuple $(\mathcal{L}, Q, \mathcal{N}, \delta)$ where:

- $\mathcal{L}$ is the set of *cells* generally taken as a subset of $\mathbb{Z}^{dim}$, dim is the dimension of the space.
- Q is a finite set of *states*. Each cell $c \in \mathcal{L}$ is associated with a value $q_c \in Q$.
- Each cell c is associated with a set of cells $\mathcal{N}(c) \subset \mathcal{L}$ called the *neighborhood* of c. The relationship $\mathcal{N}$ expresses the locality of interactions, *i.e.*, $\mathcal{N}(c)$ is constituted of cells "close" to c.
- The *local transition function* δ returns a value in Q that depends on the current state q_c and on the states of the cells in the neighborhood $\mathcal{N}(c)$.

The generation of a transactional CA consists in defining these four elements using the different components of the MAS specification. For the sake of simplicity, we restrict ourselves to MAS where:

- the set $\mathcal{P}$ corresponds to a 2D regular square grid ($dim = 2$);
- an exclusion principle holds, that limits to one the maximum number of agents on a given patch;
- the set I_a is always a singleton, that means that an agent decides to attempt only one action at each time step. Note that this case is somehow general, because any set of influences could be rewritten in only one influence: if the agent does nothing (*i.e.*, $I_a = \emptyset$), we consider that it releases the special Skip influence, and if there are more than two influences (*e.g.*, $I_a = \{\mathsf{Depose}, \mathsf{Diffuse}(d)\}$), new symbols are considered in $\mathcal{I}$ to capture the corresponding action (*e.g.*, $I_a = \mathsf{DeposeDiffuse}(d)$);
- the combination of influences $\prod I_\rho$ is always a singleton. In other words, only one action could affect a position ρ.

As shown on Figure 1, the definition of the transition function can be divided into three sub-functions δ_0, δ_1 and δ_2 corresponding to the three steps of transactional CA. As a consequence, the set of states Q can also be partitioned into three subsets Q_0, Q_1, Q_2, depending on the next step of the transactional CA to be computed.

In the following paragraphs, we briefly detail the key points of the transactional CA generation. We illustrate this translation on the example of the DLA.

Cells and Neighborhood. The set of cells exactly corresponds to the 2D grid defined by the set of patches, so $\mathcal{L} = \mathcal{P}$. The neighborhood relationship $\mathcal{N}$ is defined in such a way that $\mathcal{N}(c)$ gathers the cells that an agent a, localized on c, may access to compute its perceptions Γ_a and its actions I_a.

As an example, the particles of the DLA model may remain on the same position, and perceive or move to the positions following the North, South, West and East directions. As a consequence, the corresponding CA neighborhood relationship $\mathcal{N}$ corresponds to the classical von Neumann neighborhood; for each cell $c \in \mathcal{L}$:

$$\mathcal{N}(c) = \{c' \in \mathcal{L}, ||c - c'|| \leq 1\}$$

where $||c - c'||$ denotes the graph distance between two cells.

The Initial States Set Q_0. The set Q_0 corresponds to the initial states of a cell before three steps of the transactional CA. Let c be a cell and ρ its corresponding patch. The state of c is characterized at a given time t by

- the environment state σ_ρ^t at ρ, and
- whether there is an agent a with state σ_a^t on ρ.

Let $\widetilde{\Sigma}_{\mathcal{A}}$ denote the set $\Sigma_{\mathcal{A}} \cup \{\mathsf{Empty}\}$, where Empty represents the absence of agent. Then, the state of Q_0 are couples $(\sigma_a^t, \sigma_\rho^t)$:

$$(\sigma_a^t, \sigma_\rho^t) \in Q_0 = \widetilde{\Sigma}_{\mathcal{A}} \times \Sigma_{\mathcal{E}}$$

For the DLA model, we have $\Sigma_{\mathcal{A}} = \{\mathsf{Mobile}, \mathsf{Fixed}\}$ and $\Sigma_{\mathcal{E}} = \emptyset$: the cells of the

environment are here *passive* and holds no information. So we have:

$$Q_0 = \{\mathsf{Mobile}, \mathsf{Fixed}, \mathsf{Empty}\}$$

that corresponds to the states M_0, F_0 and E_0 of the Figure 1.

The Request Step and the Set Q_1. Compared to the elements of Q_0, the states composing Q_1 are characterized by an additional information: the influence chosen with respect to the specification of f_a (see eq. 2). In the case of an Empty cell, the particular Skip influence is used. Considering Equation 2 notations, the transition function δ_0 is defined by:

$$\begin{array}{lccl} \delta_0: & Q_0 & \longrightarrow & Q_1 = Q_0 \times I \\ & (\sigma_a^t, \sigma_\rho^t) & \longmapsto & \left\{ \begin{array}{ll} (\sigma_a^t, \sigma_\rho^t, \mathsf{Skip}) & \text{if } \sigma_a^t = \mathsf{Empty} \\ (\sigma_a^{t+1}, \sigma_\rho^t, I_a) & \text{otherwise} \end{array} \right. \end{array}$$

Note that the evaluation of the transition function δ_0 depends on the neighborhood state as it requires the perceptions Γ_a to compute $\langle \sigma_a^{t+1}, I_a \rangle$ using f_a.

In the DLA model, the set of particle actions is $I = \{\mathsf{Diffuse}(d), \mathsf{Aggregate}, \mathsf{Stay}\}$. If we identify the neutral Skip action to Stay, the definition of δ_0, based on the use of Equation 1, gives:

$$\begin{array}{lcl} \delta_0(\mathsf{Empty}) & = & (\mathsf{Empty}, \mathsf{Skip}) \\ \delta_0(\mathsf{Fixed}) & = & (\mathsf{Fixed}, \mathsf{Skip}) \\ \delta_0(\mathsf{Mobile}) & = & \left\{ \begin{array}{ll} (\mathsf{Fixed}, \mathsf{Aggregate}) & \\ (\mathsf{Mobile}, \mathsf{Diffuse}(d)) & \text{w.r.t. eq. 1} \\ (\mathsf{Mobile}, \mathsf{Stay}) & \end{array} \right. \end{array}$$

These transitions correspond to the five arrows of the "Request" column of the Figure 1. Note that some states of Q_1 are meaningless (*e.g.*, the state $(\mathsf{Fixed}, \mathsf{Diffuse}(d))$ would correspond to a diffusing fixed particle).

The Approval Step and the Set Q_2. During this step, each cell c, associated with a patch ρ, computes the set of influences I_ρ that may affect its state. This computation is done by reading the third element of the states of the neighbor cells. Then, the operator $\prod$ combines these influences into a single influence that will be taken into account during the transaction step. As a consequence, states from Q_2 refer to an additional information: the combined influence $\prod I_\rho$. The transition δ_1 is then defined by:

$$\begin{array}{lccl} \delta_1: & Q_1 & \longrightarrow & Q_2 = Q_1 \times I \\ & (\sigma_a^{t+1}, \sigma_\rho^t, I_a) & \longmapsto & (\sigma_a^{t+1}, \sigma_\rho^t, I_a, \prod I_\rho) \end{array}$$

The exclusion principle of the DLA model is specified by the definition of the operator $\prod$. Formally, let ρ be a patch, I_ρ the set of actions that affect the state of ρ, and $|S|$ the cardinal of a finite set S, we have:

$$\begin{array}{lcl} \delta_1(\mathsf{Empty}, \mathsf{Skip}) & = & \left\{ \begin{array}{ll} (\mathsf{Empty}, \mathsf{Skip}, I_\rho) & \text{if } |I_\rho| = 1 \\ (\mathsf{Empty}, \mathsf{Skip}, \mathsf{Skip}) & \text{otherwise} \end{array} \right. \\ \delta_1(\sigma_a, I_a) & = & (\sigma_a, I_a, \mathsf{Skip}) \end{array}$$

The state $(\mathsf{Empty}, \mathsf{Skip}, I_\rho)$ corresponds to the state $\mathsf{R}(d')$ of Figure 1: this state is only reachable from an empty patch with exactly one request of move.

The Transaction Step. The final step consists in computing a new state of Q_0 as a function of a state of Q_2:

$$\delta_2 : Q_2 \longrightarrow Q_0$$

The transition δ_2 computes the new state of the patch ρ and the eventual move of agents from or to ρ. This computation relies on two pieces of information:

- the influence $\prod I_\rho$ that *has to be realized*, and
- the influence I_a that is attempted.

As we assume an exclusion principle, we describe separately the case where the cell is empty and the case where an agent a is localized on the cell. If the cell is empty, δ_2 is defined by:

$$\delta_2(\mathsf{Empty}, \sigma_\rho^t, I_a, \prod I_\rho) = \left\{ \begin{array}{ll} (\sigma_a^{t+1}, \sigma_\rho^{t+1}) & (1) \\ (\mathsf{Empty}, \sigma_\rho^{t+1}) & (2) \end{array} \right.$$

where (1) and (2) correspond to two possible cases:

Case (1): $\prod I_\rho$ expresses that ρ is a target cell for an agent a and that this move has to be realized. The state σ_a^{t+1} of this agent comes from the state of the source cell. On Figure 1, the transition from state $\mathsf{R}_2(d')$ to state M_0 illustrates this case.

Case (2): $\prod I_\rho$ does not allow any agent move to the patch ρ. On Figure 1, the transition from state E_2 to state E_0 illustrates this case.

In both cases, the state of the environment may be affected by the influence $\prod I_\rho$. The new σ_ρ^{t+1} is computed using Equation 3.

If an agent a is localized on the cell, δ_2 is defined by:

$$\delta_2(\sigma_a^{t+1}, \sigma_\rho^t, I_a, \prod I_\rho) = \left\{ \begin{array}{ll} (\mathsf{Empty}, \sigma_\rho^{t+1}) & (3) \\ (\sigma_a^{t+1}, \sigma_\rho^{t+1}) & (4) \end{array} \right.$$

where (3) and (4) correspond to two possible cases:

Case (3): I_a expresses a move of the agent a from the patch ρ to another patch ρ' where $I_a = \prod I_{\rho'}$ (*i.e.*, the action is allowed by $\prod I_{\rho'}$). The patch ρ becomes empty. On Figure 1, the transition from state $\mathsf{M}_2\mathsf{D}(d)$ to state E_0 illustrates this case.

Case (4): I_a is not allowed by the neighborhood. On Figure 1, the transition from state $\mathsf{M}_2\mathsf{D}(d)$ to state M_0 illustrates this case.

In both cases, the new σ_ρ^{t+1} is computed using Equation 3.

Figure 3: Transactional CA simulation of a virtual amoebae gathering model. Under some environmental conditions, amoebae (in black) release a morphogen (whose concentration is displayed as a gray scale). Then, they follow the gradient generated by reaction-diffusion of that morphogen, until they all gather. Here, each environment patch contains up to two amoebae.

5 CONCLUSION

The transactional CA we proposed is an original solution designed to translate reactive MAS into CA. We have shown the interest of such an approach on a diffusion-limited aggregation model allowing to find the three steps required to synchronously run this MAS in a CA framework. Then we proposed the first step of a generalization and an automation of this methodology. This work is based on the use of the influence-reaction model that is naturally related to the three-step approach of transactional CA.

Other solutions have already been proposed. For example, specific kinds of CA have been designed to model the movement of particles, like the *dimer cellular automata* that develops an asynchronous point of view of the dynamics, or the *lattice gas cellular automata* initially developed for simulating fluids (Deutsch and Dormann, 2005). The question of coding moving objects in CA also led authors (Hochberger et al., 1999) to consider a two-step CA that prevents collisions. This approach is quite similar to a transactional CA. These solutions focus on the displacement of objects and have not been used yet in a more general context.

By contrast, transactional CA are developed in order to consider any kind of actions. For example, we are currently applying the methodology presented in section 4, to generate a CA corresponding to a model of gathering virtual amoebae (Resnick, 1994), an illustrative MAS that involves an active environment together with a weak exclusion principle. As shown on Figure 3, this result is promising and encourages us to use our approach on a broader range of problems.

REFERENCES

Adamatzky, A. (2001). *Computing in Nonlinear Media and Automata Collectives*. Institute of Physics Publishing.

Brooks, R. (1990). Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1-2):3–15.

Chopard, B. and Droz, M. (2005). *Cellular Automata Modeling of Physical Systems*. Collection Alea-Saclay: Monographs and Texts in Statistical Physics. Cambridge University Press.

Deutsch, A. and Dormann, S. (2005). *Cellular Automaton Modeling of Biological Pattern Formation Characterization, Applications, and Analysis*, volume 26 of *Modeling and Simulation in Science, Engineering and Technology*. Birkhäuser.

Fatès, N. (2008). Fiatlux CA simulator in Java. `http://nazim.fates.free.fr`.

Ferber, J. (1999). *Multi-Agent Systems, an Introduction to Distributed Artificial Intelligence*. Addison-Wesley.

Ferber, J. and Muller, J.-P. (1996). Influences and reaction : a model of situated multiagent systems. In *Proceedings of the 2nd International Conference on Multi-agent Systems*, pages 72–79.

Halbach, M. and Hoffmann, R. (2004). Implementing cellular automata in fpga logic. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th*, page 258.

Hochberger, C., Hoffman, R., and Waldschmidt, S. (1999). Cdl++ for the description of moving objects in cellular automata. In *PaCT'99: Proceedings of the 5th International Conference on Parallel Computing Technologies*, pages 428–435, London, UK. Springer-Verlag.

Michel, F., Beurier, G., Gouaîch, A., and Ferber, J. (2003). The turtlekit platform : Application to multi-level emergence. In *ABS 4 Agent-Based Simulation 4*.

Permalla, K. S. and Aaby, B. G. (2008). Data parallel execution challenges and runtime performance of agent simulations on gpus. In *Proceedings of the Spring Simulation Multi-Conference*, Ottawa, Canada.

Regelous, S. (2004). MASSIVE: Multiple agent simulation system in virtual environnement, http://www.massivesoftware.com/.

Resnick, M. (1994). *Turtles, termites, and traffic jams: explorations in massively parallel microworlds*. MIT Press, Cambridge, MA, USA.

Witten, T. and Sanders, L. (1981). Diffusion-limited aggregation, a kinetic critical phenomena. *Phys. Rev. Lett.*, 47(19):1400–1403.